Escola Politècnica Superior
d'Enginyeria de Manresa

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Posicionamiento en interiores a través de puntos de accesos

Grado en Ingeniería de Sistemas TIC
Curso 16/17

Autor: Cristhian Martin Campoverde Schrader
Director: Dr. Francisco del Águila López
Manresa, julio de 2017

1. RESUMEN

Este proyecto consiste en el desarrollo de un sistema que permita la localización de un terminal usando puntos de accesos.

El proyecto se ha centrado en la captura de señales de fuerza en cada sección de un plano determinado, las cuales se han usado para el posterior cálculo de distancia de un terminal respecto a los puntos de accesos y su posicionamiento en el plano.

En el documento se describirán los detalles del proceso que se ha seguido para obtener el resultado deseado, así como los obstáculos que se han presentado.

Finalmente se sacan conclusiones respecto al proyecto hecho, así como posibles ventajas/desventajas de implementar un sistema parecido pero a gran escala.

2. ABSTRACT

This project is about the development of a system which allows a device to be located using Access Point.

The project obtains signal strength in each section of a certain plane and uses it to calculate the distance between the device and the Access Points to determine its localization in the plane.

This document will detail the process followed to obtain the desired result, as well as the obstacles encountered.

Finally, conclusions will be drawn from the project, as well as possible advantages/disadvantages of implementing a similar system but on a large scale.

3. INDEX

1. RESUMEN **1**

2. ABSTRACT **2**

3. INDEX **3**

4. ANTECEDENTES **5**

5. INTRODUCCIÓN **6**

6. OBJETIVOS **7**

7. ALCANCE **8**

8. FASES DEL PROYECTO **9**

9. ESTUDIO PREVIO **10**

9.1. TECNOLOGÍA WIFI 11

9.2. PUNTO DE ACCESO Y ROUTER 11

9.3. ALGORITMOS DE LOCALIZACIÓN 12

9.3.1. TOA/TDOA (Time [difference] of Arrival) 12

9.3.2. AOA (Angle of arrival) 13

9.3.3. RSSI (Received Signal Strength Indication) 13

9.4. TÉCNICAS DE LOCALIZACIÓN 13

9.4.1. SignPost 14

9.4.2. Triangulación 14

9.4.3. Trilateración 14

9.4.4. Multilateración 14

9.4.5. Fingerprint 14

9.4.6. Probabilidad 14

9.5. PROGRAMACIÓN 15

9.5.1. PYTHON 16

9.5.2. FRAMEWORK IONIC 2 16

9.5.3. TYPESCRIPT 16

9.6. SERVIDOR CLOUD9 16

9.7. PLANO 17

10. DISEÑO **19**

11. IMPLEMENTACIÓN **24**

11.1. ORDENADOR 24

11.2. SERVIDOR 29

11.3. APLICACIÓN 34

12. VERIFICACIÓN **42**

13. CONCLUSIONES **45**

14. BIBLIOGRAFÍA **47**

4. ANTECEDENTES

Este tema no fue algo que haya decidido hacerlo desde el principio, este tema nació después de hablar con mi tutor del trabajo, el Dr. Francisco del Águila López, respecto a diferentes posibles temas para mi TFG.

Después de hablar e investigar sobre distintos temas, como hacer una red social o montar un sistema que una vez tocado el timbre de la puerta alerte al usuario mediante llamada/videollamada en un terminal (pulsera, móvil, entre otros) sobre la llegada de esta persona, y tras juntar un poco la información de ambos temas mi tutor me propuso un sistema que permita ubicar a una persona, como lo hace un GPS, pero en interiores. Después de investigar un poco respecto al tema, acepté la propuesta.

Aunque no es la primera vez que se realice este TFG en la universidad, he buscado una manera diferente de poder realizar este sistema; una manera fácil de implementar usando el conocimiento proporcionado por la universidad a lo largo del Grado cursado.

5. INTRODUCCIÓN

Actualmente el uso de tecnologías está integrado en el día a día de cada uno. Aplicaciones comunes como Facebook, Whatsapp, Maps entre otros requieren diferentes tipos de permisos para poder ejecutarse correctamente y brindar un servicio satisfactorio. De entre ellos, uno de los permisos más comunes que se requiere es la localización del dispositivo, y eso conlleva el uso del GPS.

El GPS es una tecnología usada diariamente en aplicaciones que brindan distintos tipos de servicios que requieran constancia de la situación real, ya sea de ayuda como servicios de atención personalizada, o casos simples como publicar tu ubicación en alguna aplicación social.

Si bien es cierto que el GPS es una tecnología muy usada, está limitado. Esto es debido a que hacen uso de satélites en espacios abiertos y, en espacios cerrados, como en edificios, es más difícil tener una precisión exacta.

Gracias a la evolución e implantación de tecnologías inalámbricas casos como estos se pueden resolver. Esto hace posible la implementación de un sistema de localización basado en este tipo de tecnologías.

Como los edificios hoy en día disponen de una red inalámbrica, se puede aprovechar la infraestructura conocida e implementar un sistema que incluso podría beneficiar a dicha institución:
-Los hospitales podrían conocer la posición exacta del trabajador para diferentes acciones como delegar al médico más cercano a una acción que requiera urgencia, requerir datos del paciente y obtenerlos en la pantalla de la habitación solo sabiendo la ubicación del médico, entre otros.
-Las grandes instituciones podrían implementar una aplicación que guíe a un invitado al lugar indicado (misma función del mapa de infraestructura, pero solo con una aplicación que indique ir de un punto a otro guiando a la persona).
-Los parkings podrían implementar un sistema que permita al usuario recordar el lugar donde ha aparcado su coche (en caso de que se olvide de la planta) y guiarlos hasta él.

En este proyecto se buscará una manera de brindar este tipo de servicio usando puntos de accesos dentro de un plano determinado. Se comparará distintos tipos de sistemas que puedan surgir tras un estudio previo y se escogerá el más adecuado. El resultado conllevará el desarrollo de una aplicación móvil que ponga en uso los datos obtenidos y nos brinde la localización en interiores que se quiere implementar.

6. OBJETIVOS

El objetivo principal es implementar un sistema que permita recopilar datos de distintos puntos de accesos situados en un mismo edificio.

Estos datos se almacenarán en un servidor externo y se utilizarán para ubicar un terminal en un plano determinado.

Los datos recopilados no serán reemplazados a menos que la infraestructura del edificio/plano utilizado tenga un cambio importante.

Este sistema también ha de poder diferenciar entre los puntos de acceso que pertenecen al plano definido.

Finalmente, se realizarán unas conclusiones respecto al proyecto.

7. ALCANCE

El alcance del proyecto se centra en desarrollar una aplicación móvil que hará uso de los datos obtenidos por el sistema implementado y representará la ubicación de la persona en un determinado plano.

8. FASES DEL PROYECTO

El proyecto consistirá de 4 fases importantes, las cuales se dividirán en más para ir explicando cosas paso a paso. Estas fases estarán compuestas por los pasos que se muestran en la ***Figura 1***.

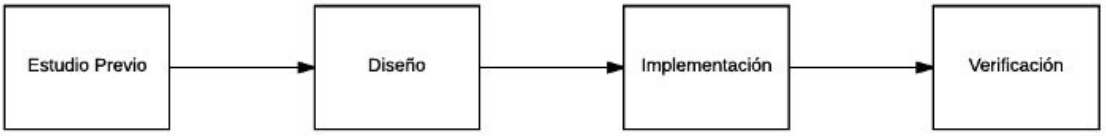

Figura 1. Fases del Proyecto

La fase de estudio previo consistirá en la recaudación de toda la información importante respecto al tema a desarrollar. Esta fase acabará descartando los posibles sistemas que se hayan planteado hasta quedarnos con uno.

La fase de diseño consistirá en planificar el desarrollo del sistema a implementar de la idea que se acaba de elegir. Se decidirá el plano donde se va a trabajar, así como el lenguaje de programación a usar, el servidor para guardar datos y la plataforma utilizada para la aplicación móvil.

La fase de implementación explicará todo lo realiza durante la puesta en marcha del sistema. Es decir, se explicará el código a utilizar en cada campo requerido, como los datos se recopilan y para qué se usarán, así como la aplicación desarrollada.

Por último, pero no menos importante, la fase de verificación cerciorará que todo lo hecho previamente funcione correctamente.

9. ESTUDIO PREVIO

La localización de un dispositivo se puede realizar de muchas formas. La manera más común es usando el GPS, pero en este proyecto la localización se realizará en espacios cerrados por lo cual primero se ha de definir qué se necesita saber.

Si se quiere saber la posición de un dispositivo, lo primero que pensamos es tener 3 o más puntos de los cuales se calcula la distancia respecto al dispositivo por cada punto y se triangula su posición. Pero, ¿Qué tecnología se usará para la localización de este dispositivo?

Existen distintos tipos de tecnologías como Wi-fi, Zigbee, Infrarrojos, RFID (Radio Frequency Identification) entre otros. De estas tecnologías trabajaremos con lo más conocido por todos: Wi-fi. Esta tecnología está presente en la vida cotidiana lo que hace su uso fácil y sin coste alguno.

Sabiendo que se trabajará con la tecnología Wifi, ahora debemos plantearnos cómo trabajaremos para la captura de datos. Es decir, ¿Qué algoritmo de localización se aplicará?. Esto es importante ya que dependiendo de uno u otro método el nivel de calibración será diferente. Por eso después de investigar un poco se han encontrado 3 tipos de algoritmos a usar:
-TOA/TDOA (Time [difference] of Arrival)
-AOA (Angle of arrival)
-RSSI (Received Signal Strength Indication)

Estos algoritmos emplean distintas técnicas de localización como:
-SignPost.
-Triangulación.
-Trilateración.
-Multilateración.
-Fingerprint.
-Probabilidad.

Estas técnicas son importantes ya que buscan maximizar la precisión en base a los datos disponibles e incluyen mecanismos para combatir el problema fundamental: el efecto Multipath (multicamino).

Aprovechando que se trabajará con la tecnología Wifi, haremos uso pleno de los Puntos de Acceso (AP). Un AP no es lo mismo que un Router; esto se explicará más adelante detalladamente.

Una vez decidido el algoritmo y la técnica de localización se ha de decidir con qué lenguaje de programación se trabajará. Como uno de los objetivos finales es hacer una aplicación

móvil, esto implicará que tanto el código que realiza las pruebas como el código que se ejecuta en la aplicación móvil sean diferentes. Esto también implica la creación, o uso, de un servidor que utilizará para procesar la información de ambos bandos .

Sabiendo que se usarán 2 lenguajes de programación diferentes para los códigos, así como un servidor, se ha de investigar también qué plano utilizar. Esto es importante debido a que si el plano es muy grande implicará que la calibración de los datos procesados será importante a la hora de devolver el resultado final.

Después de buscar todo lo que se necesitará para este proyecto, vamos a definir un poco más cada punto.

9.1. TECNOLOGÍA WIFI

Wifi es una tecnología para WLAN (Wireless Local Area Network) con dispositivos basados en el estándar IEEE 802.11. Esta tecnología surgió por la necesidad de establecer un mecanismo de conexión inalámbrica que fuese compatible entre distintos dispositivos y que permita la conexión a Internet a través de un Punto de Acceso.

Esta tecnología es la más usada, por lo que facilita el avance del proyecto ya que se aprovecha su libre disposición; aunque uno de sus puntos débiles puede ser que, a pesar de estar presente en terminales móviles, tiene una alta consumición de la batería.

9.2. PUNTO DE ACCESO Y ROUTER

AP (Access Point) y Router no significan lo mismo. Un Router es un dispositivo para la interconexión de redes que permite el enrutamiento de paquetes entre redes y provee de Internet a una vivienda. Un Punto de Acceso es un dispositivo, conectado a un Router, que crea una red inalámbrica (WLAN) en una área designada y permite la conexión de dispositivos inalámbricos a esta red, delegando el enrutamiento de paquetes al Router.

Un Router puede incorporar un Punto de Acceso, es por eso que muchas veces se confunde estos términos.

En la ***Figura 2*** nos podemos dar una idea de cómo sería un esquema de equipos que forman la red en una vivienda si el Router no tuviese el AP incorporado.

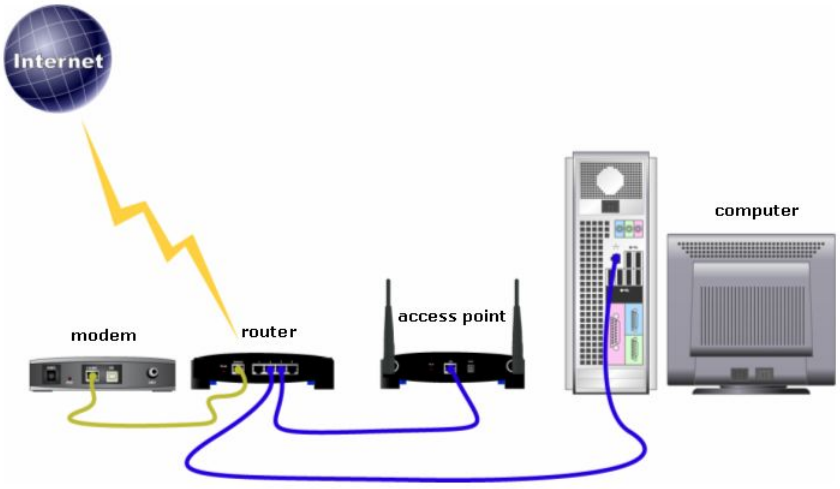

Figura 2. Esquema de conexión de dispositivos que forman una red.

9.3. ALGORITMOS DE LOCALIZACIÓN

Los algoritmos de localización son un punto importante a tener en cuenta ya que se buscará maximizar la precisión en base a los datos disponibles. Es por eso que se definirán los algoritmos a utilizar, en donde también se mostrarán las ventajas y desventajas de dicho algoritmo.

La ***Figura 3*** nos muestra un ejemplo de los 3 tipos de algoritmos a definir.

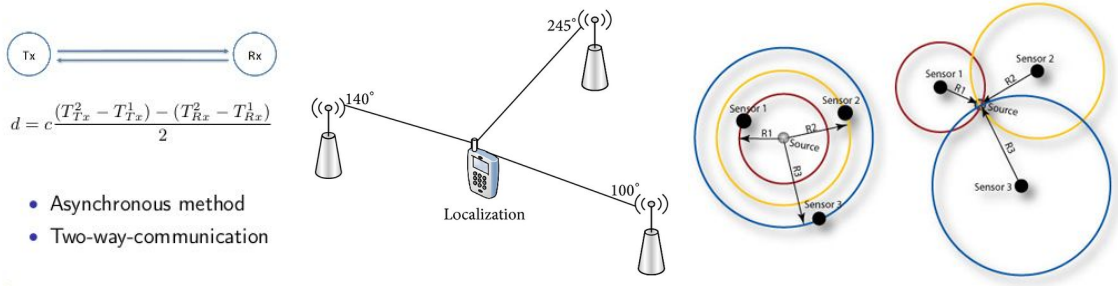

Figura 3. Ejemplos de los algoritmos TOA/AOA/RSSI respectivamente.

9.3.1. TOA/TDOA (Time [difference] of Arrival)

Este algoritmo se basa en el cálculo de la distancia entre los sensores y el terminal mediante la diferencia de los tiempos de Recepción (Rx) y Transmisión (Tx) (tiempo de propagación entre transmisor y receptor), estimando así la posición relativa del receptor.

El tiempo de propagación es el que necesita la señal para viajar desde el transmisor al

receptor y puede calcularse como (t. transmisor – t. receptor). A partir de este tiempo y sabiendo que la velocidad de propagación es aproximadamente la de la luz se obtiene la distancia entre transmisor y receptor.

Esta técnica tiene dos problemas básicos, el primero de ellos es que no haya una trayectoria directa entre el transmisor y el receptor, la segunda es que los relojes de ambos dispositivos deben estar sincronizados para obtener una distancia correcta, lo cual afecta en el coste de los dispositivos.

9.3.2. AOA (Angle of arrival)

Este algoritmo se basa en el ángulo de llegada de la señal desde el punto de vista del terminal usado. El ángulo de llegada se calcula trazando una línea entre el receptor y emisor y otra desde el receptor hacia una dirección conocida como pueden ser los puntos cardinales. Usando varios receptores en posiciones conocidas se puede determinar la posición del transmisor mediante triangulación.

La precisión de este aumenta según el número de antenas utilizadas y también a su vez el coste.

No es una buena opción en redes Wifi debido a que este algoritmo es efectivo cuando no existe multicamino ya que las señales reflejadas pueden provocar una desviación del ángulo de llegada.

9.3.3. RSSI (Received Signal Strength Indication)

Este algoritmo se basa en la potencia de señal recibida para obtener una estimación de la distancia. Debido a problemas como la reflexión, refracción, difracción y dispersión de las señales sobre objetos, la relación entre la potencia y la distancia puede variar demasiado. Por eso, para poder mejorar la relación entre distancia y potencia, se hará uso de técnicas de localización. Aún así, este algoritmo parece bastante asequible en cuanto a implementación en este proyecto.

9.4. TÉCNICAS DE LOCALIZACIÓN

Este es un punto importante a tener en cuenta ya que estas técnicas combaten el efecto Multipath. Es por eso que vamos a definir un poco de qué trata cada técnica.

9.4.1. SignPost

Esta técnica busca obtener la posición del dispositivo en base al sensor más próximo. Es como aplicar un algoritmo RSSI simple. Esta técnica puede ayudar a obtener las posibles secciones próximas dentro de un plano grande.

9.4.2. Triangulación

Esta técnica está basada en el Ángulo de llegada (AOA), pero requiere más medidas con más de un sensor. Es la técnica usualmente usada con el algoritmo AOA ya que al tener más de un dispositivo puedes triangular la posición del dispositivo mediante ángulos.

9.4.3. Trilateración

Esta técnica está basada en la estimación de distancias. Usualmente se usa con el algoritmo RSSI y TOA, en cuyo caso se requeriría la sincronización de los elementos pertenecientes a la red.

9.4.4. Multilateración

Es una técnica más precisa que Trilateración pero con más cálculos complejos. Requiere sincronizar con la red también ya que se usa con el algoritmos TDOA (TDOA tiene más precisión que TOA).

9.4.5. Fingerprint

Esta técnica requiere la calibración previa del entorno donde se encuentra desplegado la red de sensores inalámbricos (Puntos de Accesos, Raspberry, entre otros) para poder generar la base de datos Fingerprints y utilizar estos datos como base para posteriores cálculos.

9.4.6. Probabilidad

Esta técnica requiere previamente realizar pruebas. Esta solución evita calibraciones mediante la aplicación de algoritmos heurísticos que consideran el histórico de posiciones. Es decir, si ya tenemos un historial de posiciones podemos aprovecharlo.

9.5. PROGRAMACIÓN

La parte de la programación será importante ya que de él derivarán los datos necesarios para su posterior utilización. Es por ello que se ha de decidir el tipo de lenguaje de programación a usar.

A lo largo de la carrera se ha trabajado con lenguajes como Python, C, VHDL entre otros. C es una buena opción para programar, pero me he decantado por Python por la facilidad a la hora de programar gracias a la gran cantidad de librería que posee.

Pero no solo se trabajará con Python, también hay que definir el tipo de lenguaje a utilizar para la aplicación móvil. No obstante, primero se ha de elegir el Framework a utilizar ya que de él depende el tipo de lenguaje.

Un Framework es un entorno de trabajo basado en una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Por eso, la decisión de qué Framework se utilizará es importante, ya que de él depende el desarrollo de la aplicación.

No obstante, Ionic 2 será el Framework a utilizar pero... ¿Por qué este Framework?.

La decisión de utilizar este Framework se debe a mi hermano, quien programa con ello y me ha ayudado explicando lo que he de tener en cuenta a la hora de programar una aplicación móvil. Me ha dado tips y me ha enseñado a desarrollar una aplicación simple de prueba para empezar a entender la manera correcta de trabajar con ello.

En este Framework el lenguaje de programación a utilizar será TypeScript, un lenguaje orientado a objetos y bastante dinámico.

Otro punto importante a tener en cuenta es el servidor en el cual se guardarán los datos recopilados y del cual se obtendrán estos datos para utilizarlos en la aplicación. En un principio pensaba crear un servidor para hacer las pruebas pero, aprovechando que existen servidores en Internet y que estos son gratuitos, decidí utilizar un servidor online. El servidor a utilizar será Cloud9, un servidor virtual gratuito y perfecto para este proyecto.

A continuación se explicará el lenguaje, framework y servidor a utilizar.

9.5.1. PYTHON

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

Python apuesta por la simplicidad, versatilidad y rapidez de desarrollo, lo que lo hace un lenguaje perfecto para este proyecto.

9.5.2. FRAMEWORK IONIC 2

Ionic 2 es un framework para el desarrollo de aplicaciones híbridas, inicialmente pensado para móviles y tablets, aunque ahora también capaz de implementar aplicaciones web e incluso dentro de poco aplicaciones de escritorio multiplataforma. Su característica fundamental es que usa por debajo Angular 2 y una cantidad de componentes enorme, que facilita mucho el desarrollo de las aplicaciones.

Se trata de una estupenda herramienta para la creación de aplicaciones sorprendentes, pensada para obtener resultados de una manera rápida y con una menor inversión económica, ya que permite crear aplicaciones para distintas plataformas móviles con una misma base de código.

9.5.3. TYPESCRIPT

TypeScript es un lenguaje de programación de código abierto desarrollado y presentado por Microsoft hace unos tres años. Es un superconjunto de JavaScript que esencialmente añade capacidades de POO como es el tipado estático y objetos basados en clases.

Extiende la sintaxis de JavaScript, por medio de un lenguaje propio que compila ficheros en lenguaje JavaScript original, asegurando la compatibilidad con todos los navegadores, servidores y sistemas operativos.

9.6. SERVIDOR CLOUD9

Estamos en una época de florecimiento de los entornos de desarrollo en la nube y Cloud9 es una de las mejores opciones que podemos encontrar. Básicamente lo que tienes aquí es un programa que se ejecuta sobre el navegador pero que a pesar de ese limitante tiene las

funcionalidades más destacadas de un IDE en el que además tenemos un entorno de trabajo real donde podemos poner nuestros programas en ejecución, ya sean sitios web o programas ejecutables.

El sistema es gratuito y te asignan recursos suficientes para trabajar y probar tus aplicaciones en un entorno totalmente autónomo y configurable a tu gusto, en el formato de los conocidos VPS (Virtual Private Server).

Entre las ventajas de este VPS con respecto a lo que puedes conseguir con el desarrollo local encontrarás:

-No necesita configuración: Con crear tu usuario puedes disponer de tu servidor listo para usar con los lenguajes más habituales.
-Probar en un entorno más real: Puedes probar tu código en un entorno más real que a través de "localhost". En un dominio remoto, de Internet y un sistema operativo habitual en sitios en producción.
-Posibilidad de compartir con otras personas: sin que tengan que hacer nada en especial.
-Trabajar desde cualquier ordenador: con el único requisito que esté conectado a Internet y sin tener que instalar ningún software en ese ordenador.

Para empezar a trabajar con C9 revisar la documentación.

9.7. PLANO

El plano a utilizar es bastante importante al momento de tenerlo en cuenta. Un plano grande implica calibración extra en el procesamiento de datos para brindar una solución buena. Un plano pequeño implica que, aunque no haya calibración extra, los datos sean correctos en cuanto a su recopilación y tratamiento, para así poder implementar calibración extra si se desea ampliar este plano.

Es por eso que, en cuanto a este proyecto, se ha decidido que el plano a utilizar sea uno pequeño.

Después de buscar posibles planos a utilizar se ha decidido que la distribución del plano será como en la ***Figura 4***.

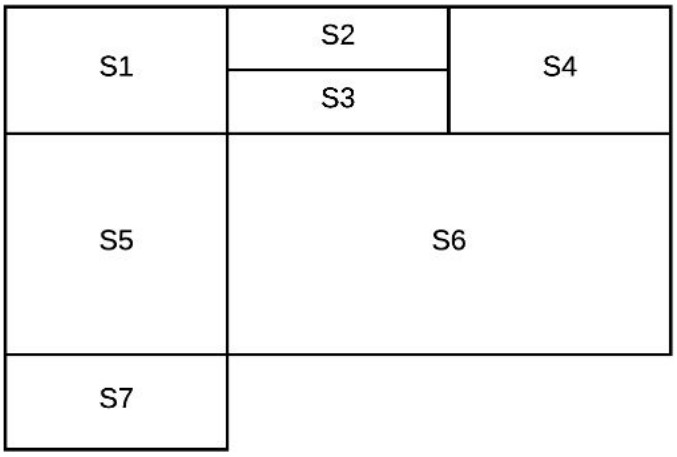

Figura 4. Distribución del plano por secciones.

10. DISEÑO

Como ya se ha visto en el estudio previo, se hará uso de la tecnología Wifi para realizar este proyecto.

Desde el punto de vista de los datos a tratar, previo a la aplicación, se ha pensado que los datos recopilados previamente se suban al Servidor para así después utilizarlos en la Aplicación. Por esa parte está claro lo que se tiene que hacer pero, ¿Cómo trabajaría la App?.

Lo ideal sería descargar los datos del servidor en la aplicación y trabajar directamente allí, pero también se puede subir los datos recopilados de la app al servidor, trabajar allí y recopilar el resultado para aplicar la acción correspondiente. Dicho esto, este sistema sería fácil de implementar e implicaría que la aplicación solo presente los resultados en un formato deseado.

Teniendo una idea de como sería el tratamiento de datos podemos decir que el diseño general sería el visto en la ***Figura 5***.

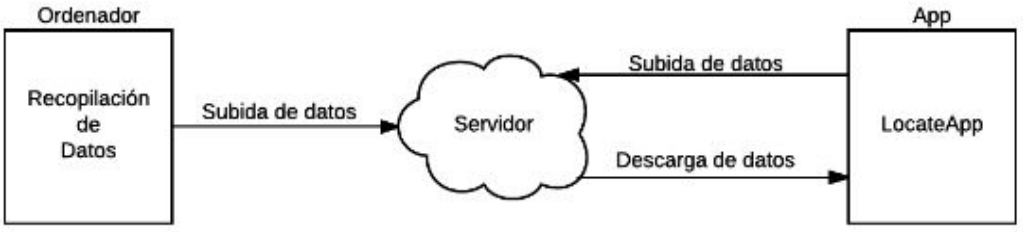

Figura 5. Diseño general de este proyecto

Una vez decidido cómo se van a tratar los datos, hemos de decidir cómo recopilamos esta información.

Como también se ha visto en el Estudio Previo, se trabajará, para ser exacto, con Puntos de Accesos debido a su disponibilidad y no implementación extra a la hora de trabajar.
Aún así, hay que tener en cuenta que estos Puntos de Accesos (AP) han de estar distribuidos dentro del plano diseñado. Por eso, y para poder trabajar de una manera cómoda, se han colocado 3 AP en 3 distintos puntos, tal y como se puede ver en la ***Figura 6***, de los cuales se recuperarán los datos.

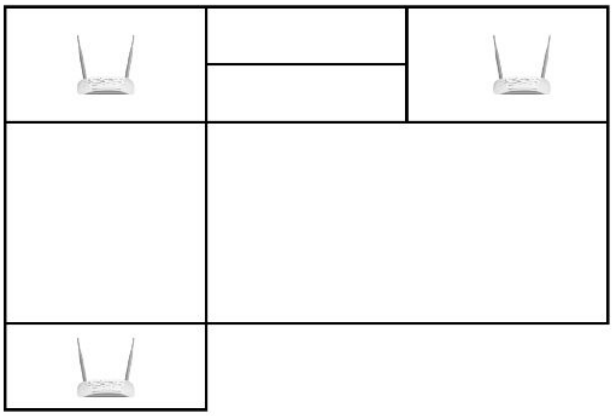

Figura 6. Distribución de AP's en el plano

Ahora toca recopilar datos. La recopilación de datos del entorno, previa a la aplicación, se realizará desde un ordenador para mayor comodidad. Por ello el código a ejecutar ha de ser capaz de diferenciar los Puntos de Accesos (AP) que no pertenecen al plano debido a que estos datos no se utilizarán. Por eso, se ha de filtrar una lista de AP con los ESSID's (nombres, no MAC). También se ha de elegir el tipo de datos a guardar ya que el código a ejecutar obtiene datos de los cuales muchos no interesan. Esto conlleva a elegir la información al momento de procesarla.

Cabe destacar que el código para realizar pruebas desde el ordenador portátil, que será la base para la aplicación móvil, será no modificable. Es decir, los datos recopilados no se cambiarán a excepción de que el edificio/plano que se ha usado inicialmente sufra un cambio a nivel estructural (una nueva habitación, cambio de posicionamientos de AP o añadir un nuevo AP).

Dicho esto, ¿Qué tipo de datos recopilamos exactamente?

Después de estudiar los 3 algoritmos presentados, así como las técnicas de localización disponibles, se ha decidido a utilizar el algoritmo RSSI con la técnica FingerPrint. Esto se debe a que una implementación del algoritmo TOA/TDOA requiere mucha precisión y depende de la sincronización de dispositivos, mientras que el algoritmo AOA requiere varios dispositivos para una precisión exacta, lo que eleva su coste. Por su parte, el algoritmo RSSI es fácil de implementar ya que la obtención de estos datos es simple de implementar gracias a los estudios vistos durante el Grado en la parte de informática.

Como se va a implementar el algoritmo RSSI, esto conlleva a recopilar datos de los diferentes puntos de accesos y quedarnos con información esencial como su ESSID, para posterior filtración con una lista de AP's, su BSSID, que es la MAC del dispositivo, y la señal de potencia que es la información que realmente nos interesa, pero que hemos de separarlo por MAC para después hacer los cálculos correctos donde corresponda.

El uso de la técnica FingerPrint se debe a que hace uso de una base de datos recopilados previamente en un entorno calibrado para el cálculo posterior de lo que se quiere obtener; en nuestro caso, la distancia para determinar la sección.

Por eso, después de crear una base de datos en donde estarán las MAC con sus correspondiente señal de potencia separados por tiempo (5min de obtención de datos), estos datos se utilizarán para calcular potencias medias por sección, separados por la potencia media vista por cada Punto de Acceso. Una vez calculado la potencia media por sección y por punto de acceso, se obtendrán nuevos datos de los cuales se usarán para calcular la distancia Euclidiana.

La distancia Euclidiana es la distancia "ordinaria" (que se mediría con una regla) entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras. La fórmula a utilizar sería:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Donde x_2 pertenece a la potencia media y x_1 pertenece a la nueva potencia calculada.

Los nuevos datos a recopilar se realizarán en la aplicación. Esto implica implementar un código que permita, de la misma manera que en un ordenador, obtener la información respecto a los Puntos de Accesos.

Sabiendo qué datos se han de recopilar, y cómo, ahora se ha de plantear cómo pasar la información al servidor, procesandola en el formato correcto. Esto implica que o bien los datos sean transmitidos directamente al momento de capturarlos o que se cree un menú para realizar diferentes acciones.

Para esto, se ha considerado mejor que los datos recopilados previamente (desde un ordenador) se suban directamente a la nube (servidor) sin necesidad de realizar un código que te permita conectar al servidor y subirlo automáticamente. Ciertamente, la idea de realizarlo mediante código es más tentadora porque evita que personalmente te conectes a un navegador, ingreses tus datos personalmente y luego subas los archivos, pero como en algún momento tienes que entrar para trabajar con tu workspace entonces puedes aprovechar ese momento y subirlo personalmente.

Respecto a los datos recopilados desde la aplicación, se ha pensado que sería mejor pasarlos al servidor mediante un *request*. Esto implica poner los datos en la *URL*, recopilarlos en el servidor, ejecutar un código que realice los cálculos necesarios y devolver la respuesta en formato de página web. La respuesta, para mayor comodidad, será el número (o texto) de la sección a la que corresponde la menor distancia, dato que después se utilizará para indicar la sección a la que el terminal se localiza en ese instante en la aplicación.

Ahora que se ha explicado cómo se diseñaría este proyecto, veremos los esquemas/diseños que se seguirían para su correcto funcionamiento.

Por parte del código a ejecutar en el ordenador, que sería código Python, el esquema sería el visto en la ***Figura 7***.

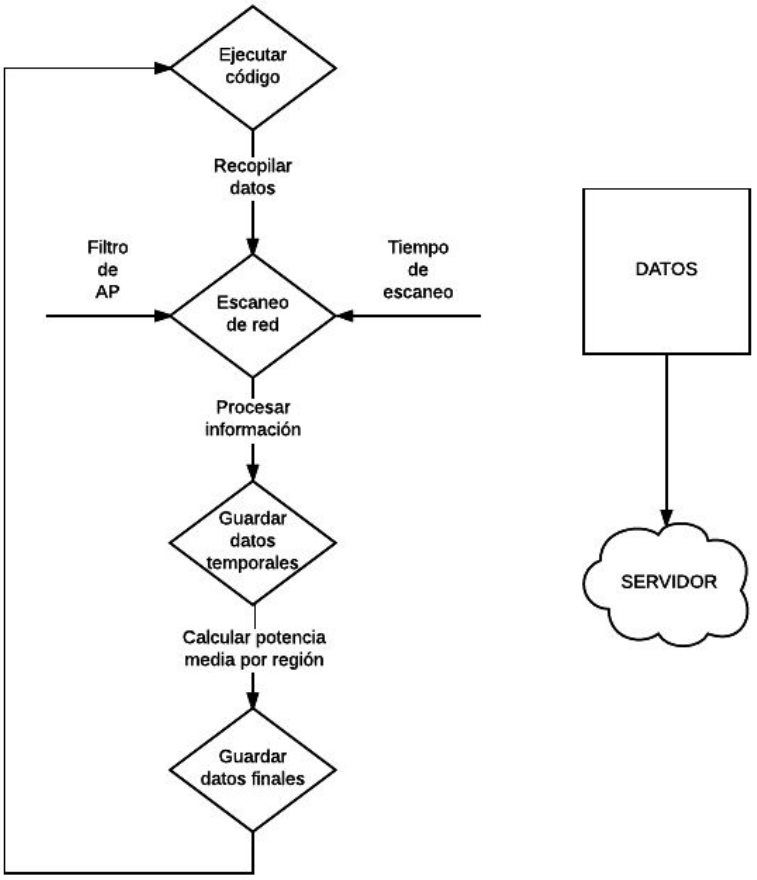

Figura 7. Diagrama del código Python a seguir.

La explicación rápida, ya sabiendo cómo se trabajará en completo según lo visto previamente, sería que la acción de recopilar datos pasaría por escanear la red pasando un filtro de AP's y con un tiempo determinado de escaneo. Luego estos datos se procesarán para obtener solo la información necesaria y después se guardarán en un archivo. Posteriormente a esto, se calculará la potencia media por sección y estos nuevos datos serán los finales, los cuales se guardarán y se subirán a la nube.

Por otro lado tenemos la aplicación y la parte del servidor.

Como la aplicación depende del servidor, se adjuntará un solo esquema y será el visto en la ***Figura 8***.

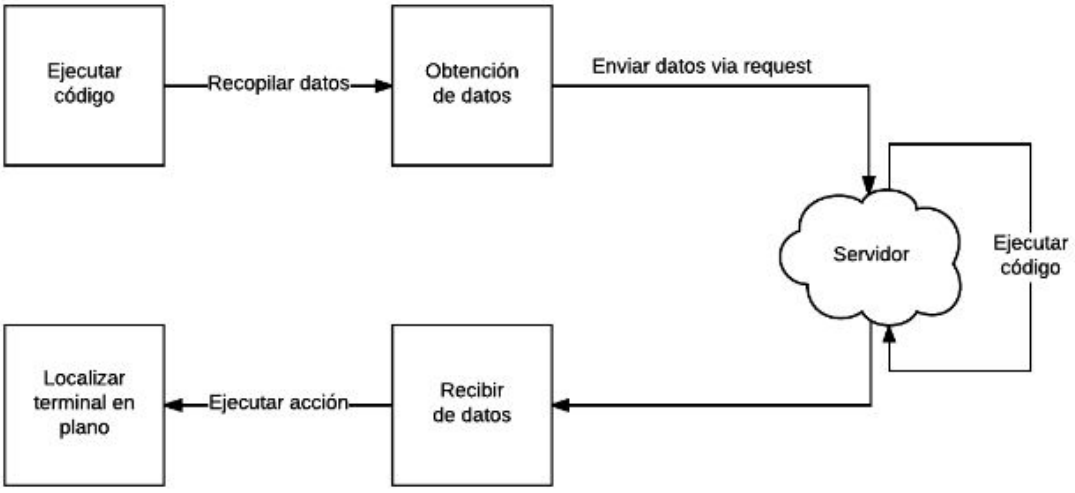

Figura 8. Diagrama del código de la aplicación a seguir junto al servidor.

La explicación rápida sería que la aplicación obtiene nuevos datos y los envía al servidor, quien los procesa de la manera correcta y devuelve una respuesta, que será utilizada por la aplicación para localizar el terminal en el plano.

Complementando lo anterior, se creará en la aplicación el plano a utilizar con sus 7 secciones de las cuales al momento de localizar el terminal se llevará a cabo una acción. Esta acción puede ser o bien encender la sección en donde se encuentra el terminal o bien utilizar un punto que haga referencia al terminal dentro de esta sección.

11. IMPLEMENTACIÓN

La implementación se dividirá en 2 partes: la implementación en el ordenador en Python, la implementación en el servidor (la ejecución del procesamiento de datos), y la implementación en para la aplicación en TypeScript.

11.1. ORDENADOR

Primero el código a implementar es el de Python que crea la base de datos FingerPrint. Mientras se creaba el código se han importado algunas librerías, así como se ha implementado la codificación UTF-8 para los caracteres como la 'ñ' o los tildes, entre otros.

De las librerías a implementar están:
-os: permite trabajar con funcionalidades pertenecientes al Sistema Operativo.
-strftime: convierte a una cadena de string la hora.
-gmtime: te da la hora global (fecha + hora completa).
-time: te da la fecha actual (en segundos, UTC).

Utilizando la librería os se ha utilizado *os.system* para ejecutar una comanda en la propia consola, tal y como se verá más adelante.

```
# -*- coding: utf-8 -*-
#!/usr/bin/python

import os
from time import strftime, gmtime, time

pw = os.system
```

Primero se ha creado el filtrado de los Puntos de Accesos a utilizar. Se define la clase *apList*, los argumentos si hacen faltan y la acción a utilizar. Esta acción leerá el archivo, obtendrá los ESSID's de los AP's y devolverá por resultado estos ESSID's.

```
def apList(apdata):
        """
        Lista de los ESSID's de los AP del entorno a filtrar
        """
        data = list()
        f = open(apdata,"r")
        for line in f:
        data.append(line.split("\n")[0])
```

```
        f.close()
        return data
```

También se define la clase *scanList*, sin argumentos en este caso. Esta clase ejecutará en la consola el escaneo de red filtrando solo los argumentos requeridos: ESSID (para compararlo después con el filtrado de AP's), Address (MAC) y Signal (la señal de potencia). Estos datos serán guardados en el archivo *scan.txt* de manera automática, para después recuperarlos y guardarlos en un diccionario.

```
def scanList():
        """
        Escaneo de la red
        """
        name = 0
        signal = 0
        addr = 0
        value = dict()
        mac = dict()
        data = list()
        pw("/sbin/iwlist wlan0 scan | egrep 'ESSID|Address|Signal' > scan.txt")
        f = open("scan.txt","r")
        for line in f:
        if "ESSID" in line:
        name = line.split('ESSID:"')[1].split('"')[0]
        elif "Signal" in line:
        signal = line.split("level=")[1].split(" dBm")[0]
        elif "Address" in line:
        addr = line.split('Address: ')[1].split('\n')[0]
        if name!=0 and signal!=0 and addr!=0:
        mac[name] = addr
        value[name] = signal
        name = 0
        signal = 0
        addr = 0
        f.close()
        pw("rm scan.txt")
        data.append(value)
        data.append(mac)
        return data
```

Posteriormente se crea la clase *dataAcquisition* con los siguientes argumentos: scanTime (el tiempo de escaneo), saveData (el archivo donde se guardarán los resultados) y apData (filtro de AP's). En esta clase se hará un escaneo del tiempo que se quiera realizar, y mientras el tiempo actual sea menor al tiempo total de escaneo se comparará los datos obtenidos por el escaneo con el filtro de AP's y los que coincidan se guardarán en el archivo deseado con el siguiente formato:

Año-Mes-Día Hora:Minutos:Segundos, 'MAC', DB

Se guardarán así para ver como varían los resultados con cada segundo.

```
def dataAcquisition(temps,savedata,apdata):
    """
    Filtra los ap y procesa solo la información deseada
    Adquiere datos según el tiempo que se desea
    """
    f = open(savedata,"w")
    t_end = time() + 60*temps
    a = apList(apdata)
    while time() < t_end:
        dataList = scanList()
        data_keys = dataList[0].keys()
        i = 0
        while True:
            if i>=len(data_keys):
                break
            elif data_keys[i] in a:
                f.write(strftime("%Y-%m-%d %H:%M:%S,'",gmtime())
                    + dataList[1][data_keys[i]] + "',"
                        + dataList[0][data_keys[i]] + "\n")
            i+=1
    f.close()
    return 1
```

Como el objetivo es crear una base de datos para el fingerprinting, es necesario realizar la media de las potencias capturadas para su posterior cálculo de la distancia Euclidiana. Por eso, se calculará la potencia media por MAC; es decir, la potencia media por cada archivo que tiene datos recopilados (se ha de realizar este proceso tantos archivos se hayan creado; respecto al archivo a guardar, ha de ser el mismo para que se puedan adjuntar los nuevos datos). Después de ello, se guardará en un nuevo archivo en el siguiente formato:

1: {'MAC1': value1, 'MAC2': value2, ...}

Donde *1* significa la sección seguida de cada MAC del Punto de Acceso con su potencia media. Pero esto no significa que *1*, que representa la sección, sea un "*número"*. Si se recopila más de un archivo en una misma sección (ya que a mayor espacio menor precisión, lo que implicaría hacer más capturas de datos en diferentes puntos de la sección para obtener una mayor precisión) éste será indicado con el formato *Sección_#Dato*. Es decir, si tenemos 4 archivos de datos recopilados en la misma sección, al momento de guardarlo se ha de realizar con el siguiente formato:

2_0: {'MAC1': value1, …}

2_1: {'MAC1': value 1, …}

Esto, obviamente, será indicado al momento de guardar el archivo.

```
def avgPotSection(saveData,section,avgData):
    """
    Potencia media por sección
    """
    a = list()
```

```
    b = list()
    avgList = list()
    f = open(saveData,"r")
    for line in f:
        if line[21:38] not in a:
            a.append(line[21:38])
            b.append(list())
        b[a.index(line[21:38])].append(line[-3:-1])
    f.close()
    # media aritmetica
    i = 0
    while i<len(b):
        j = 0
        avg = 0
        while j<len(b[i]):
            avg += int(b[i][j])
            j += 1
        avg = avg/len(b[i])
        avgList.append(avg)
        i += 1
    i = 0
    text = section + ": "
    avgDict = dict()
    while i<len(b):
        avgDict[a[i]] = int(avgList[i])
        i += 1
    text += str(avgDict) + "\n"
    f = open(avgData,"a")
    f.write(text)
    f.close()
    return 1
```

Una vez obtenido este nuevo documento, se recuperarán los datos y se ordenarán, esta vez, por *MAC*. Esto conllevará a guardar todo en una lista que contenga los diferentes diccionarios por *MAC* y por sección. El formato a guardar será el siguiente:

{'MAC1': {'1': value1, '2_1': value2, '3_2': value3, …, '7': value7}}

Se ha realizado en este formado por comodidad a la hora de procesar los datos desde el servidor, una vez recopilado los datos de la aplicación.

```
def orderAvgPot(oldData,newData):
    avg = dict()
    keys = list()
    sect = list()
    sect0 = list()
    f = open(oldData,"r")
    j = 0
    for line in f :
        k = line.index(":")+2
        section = line[:k-2]
```

```
    avg = eval(line[k:-1]) #dict {MAC: values}
    dictKeys = avg.keys()
    i = 0
    while i<len(dictKeys):
        if dictKeys[i] not in keys:
            keys.append(dictKeys[i])
            sect.append(dict())
        i += 1
    k = 0
    while k<len(keys):
        value = avg[keys[k]]
        sect[k][section] = value
        k += 1
    j += 1
    print sect
f.close()
f = open(newData,"w")
i = 0
while i<len(keys):
    text = "{'" + str(keys[i]) + "': " + str(sect[i]) + "}\n"
    f.write(text)
    i += 1
f.close()
```

Finalmente se ejecutarán las clases creadas.

A la hora de ejecutar estas clases (se ha de comentar con # las clases que no se utilizarán en su momento, como *avgPotSection* y *OrderAvgPot* ya que primero dependen de la recopilación de datos para crear la base deseada) se especifica el tipo de archivo que se requiere para que el usuario no tenga dudas.

```
apdata = raw_input("Nombre de archivo de AP's: ")
savedata = raw_input("Nombre de archivo para guardar datos: ")
loaddata = raw_input("Nombre de archivo para recuperar datos: ")
tmpdata = raw_input("Archivo temporal (mismo para todos): ")
sectiondata = raw_input("Sección del plano\nformato: #sección_#dato (+1 captura de
datos - 2_0,2_1, 2_2..; sino #sección): ")
dataAcquisition(5,savedata,apdata)
avgPotSection(loaddata, sectiondata, tmpdata)
orderAvgPot(tmpdata, savedata)
```

11.2. SERVIDOR

Los datos obtenidos, después de toda la recopilación de datos, serán subidos al servidor Cloud9. Se ha optado por subirlos personalmente debido a la falta de tiempo para completar una acción propia desde el código Python hacia el servidor. No obstante, los datos estarán a disposición del usuario de la aplicación.

Una vez subidos los datos se ejecutará el servidor Apache en Cloud9, quien proveerá de un link al cual se puede acceder desde cualquier lado siempre y cuando no se detenga el servidor:
https://cris-app-schraderleo.c9users.io/

Desde la aplicación, después de recopilar datos se enviarán estos a una página en particular de este dominio:
https://cris-app-schraderleo.c9users.io/hello.php

Debido a que los datos se enviarán via URL, el formato a enviar será el siguiente:
https://cris-app-schraderleo.c9users.io/hello.php?list=MOVISTAR_X1,D4:7B:B0:09:18:88,42,MOVISTAR_X2,98:97:D1:0D:AC:F3,40,MOVISTAR_X3,D4:7B:B0:79:48:88,49,MOVISTAR_X4,62:02:71:75:89:24,60

Los datos a enviar desde la aplicación incluirán todos los puntos de accesos y/o routers que detecte la aplicación con su respectiva MAC y señal de potencia, esto en una etiqueta llamada *list*.

Una vez llegado los datos a esta página (*hello.php*) éstos se obtendrán y se guardarán en el archivo *appData.csv*, el cual será utilizado a continuación cuando se ejecute el programa *execpy.py* de manera automática (se han convertir a ejecutable este archivo). Todo esto se realiza desde el mismo código de la página *hello.php*. Cuando el ejecutable acabe éste devolverá un valor el cual se adjuntará cuando se redireccione a la página *response.php* con *value* como etiqueta.

```
<html><body>
<?php

echo 'Hello world from Cloud9!';

$newlist = $_GET['list'];

if ($newlist) {

    $file = 'appData.csv';
    file_put_contents($file,$newlist);
```

```
    $command = escapeshellcmd('/home/ubuntu/workspace/execpy.py');
    $newvalue = shell_exec($command);

    header("Location:
https://cris-app-schraderleo.c9users.io/response.php?value=$newvalue");

}

?>
</body>
</html>
```

La página *response.php* esperará este nuevo valor y mostrará como página el valor recibido. Este valor representará el número de la sección correspondiente a la ubicación del terminal.

```
<html><body>
<?php

$newlist = $_GET['value'];
echo $newlist;

?>
</body>
</html>
```

Respecto al ejecutable *execpy.py*, abrirá los archivos *aps.csv*, *finaldata.csv* y *appData.csv* y recopilará la información para ejecutar los cálculos correctos:
-*aps.csv* contiene la lista de nombres de los Puntos de Accesos a utilizar, los cuales se usarán de filtro.
-*finaldata.csv* será el nombre del archivo final donde se guardará la base de datos por puntos de accesos y por secciones.
-*appData.csv* será el archivo donde se guardará la información recopilada desde la aplicación.

El código recopilará los nuevos datos, filtrará la lista de AP's para obtener solo la información necesaria, separará las potencias medias por AP's y secciones en una lista nueva determinada solo por secciones (si hay 3 AP's con su potencia media respectiva por sección, la nueva lista será *[[PotM1, PotM2, PotM3], [...], ...]* donde la sección 1, que corresponde al elemento 0 de la lista, tendrá todas las potencias medias de cada punto de acceso en una lista) y calculará la distancia Euclidiana con las nuevas potencias recibidas, donde finalmente ésto se depositará en una lista definitiva que contendrá las distancias calculadas.

Esta lista puede tener 2 formatos diferentes:

[Pot1, Pot2, Pot3, Pot4, Pot5, Pot6, Pot7]

[Pot1, [Pot2.1, Pot2.2, Pot2.3], Pot3, Pot4, Pot5, Pot6, Pot7]

Estos formatos se deben a que si hay más de una base de datos para la misma sección éstos se adjutarán en una lista propia que posteriormente se añadirá a la lista final. Así, tendríamos, por ejemplo, 3 distintas distancias en una misma sección, de las cuales al momento de decidir la sección a la cual se ubica el terminal (que corresponde a la distancia mínima) se calcularía la distancia mínima de esta sección que contiene más de una distancia, y luego se calcularía la distancia mínima entre todas las secciones. Dicho esto, se devuelve la sección correspondiente (*1,2,3,4,...*) y ésta se utilizará en *response.php* tal y como se ha explicado antes.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import math

"""
Obtener datos de archivos
"""

f = open("aps.csv","r")
apData = f.readlines()
f.close()

f = open("finaldata.csv","r")
oldData = f.readlines()
f.close()

f = open("appData.csv","r")
newData = f.readlines()
f.close()

"""
Trabajar con appData
Nuevos datos de la app
Filtrar aps
"""

dictName = dict();
name = ""
mac = ""
value = 0
i = 0
dataSplit = newData[0].split(",")
while i<len(newData[0].split(",")):
    if name=="":
        name = dataSplit[i]
        i+=1
    elif mac=="":
```

```
        mac = dataSplit[i]
        i+=1
    elif value==0:
        value = int(dataSplit[i])
        i+=1
    if name!="" and mac!="" and value!=0:
        dictMAC = dict()
        dictMAC[mac] = value
        dictName[name] = dictMAC
        name = ""
        mac = ""
        value = 0

newDict = dict()
j = 0
while j<len(apData):
    data = apData[j].split('\n')[0]
    if data in dictName.keys():
        newDict.update(dictName[data])
    j+=1

"""
Trabajar con base de datos
Datos ya recopilados
"""

sectionList = list()
oldDict = dict()
dist = dict()
list0 = list()
list1 = list()
list2 = list()
k = 0
#Recupera diccionario de archivo
while k<len(oldData):
    oldDict.update(eval(oldData[k]))
    k+=1
#Recupera llaves del diccionario (APs)
i = 0
while i<len(newDict.keys()):
    keys = newDict.keys()[i]
    list0.append(oldDict[keys])
    i+=1
#Recupera llaves del diccionario dentro de APs (secciones)
key0 = list0[0].keys()
#Recupera potencias medias de las secciones
i=0
while i<len(key0):
    j=0
    listX = list()
```

```
    while j<len(list0):
        listX.append(list0[j][key0[i]])
        j+=1
    list1.append(listX)
    i+=1
k = 0
#Recupera valores del diccionario de nuevos datos
values = newDict.values()
#Obtiene distancia euclidiana entre potencias nuevas y viejas
while k<len(list1):
    j = 0
    section = 0
    while j<len(list1[k]):
        section += (values[k]-list1[k][j])**2
        j+=1
    list2.append(math.sqrt(section))
    k+=1
#Revisa lista de secciones
#Si encuentra una sección con +1 potencia se calcula la media (2_0,2_1,...)
#Adjunta medias por secciones
tmpList = list()
i = 0
while i<len(key0):
    tmp = key0[i].split("_")
    if len(tmp)>1:
        tmpList.append(list2[i])
    else:
        if len(tmpList)>0:
            sectionList.append(tmpList)
            tmpList = []
        sectionList.append(list2[i])
    i+=1
if len(tmpList)>0:
    sectionList.append(tmpList)
#Lista de potencias medias por secciones
#Por distancia minima (en caso de +1 potencia en la misma sección)
finalSectionList = list()
j = 0
while j<len(sectionList):
    if type(sectionList[j]) is list:
        finalSectionList.append(min(sectionList[j]))
    else:
        finalSectionList.append(sectionList[j])
    j+=1
print list2.index(min(finalSectionList))+1
```

11.3. APLICACIÓN

Por parte de la aplicación, el código ha estado dividido de la siguiente manera:

Primero hay que instalar Ionic 2. Para instalar Ionic 2 hay varios componentes que se necesitan como AndroidStudio. Para no entrar tan a detalle en su instalación, seguir los pasos de su instalación en el link propiciado en la bibliografía.

Una vez instalado Ionic 2, crear la aplicación dentro del proyecto *Myapp* (que se ha de crear) dentro de la carpeta AndroidStudioProjects ubicado en *$HOME*. Como es la primera vez que se crea una aplicación, se utilizará la opción *tutorial* para aprovechar la estructura básica de la aplicación que luego se modificará.

La comanda a utilizar será:
$ ionic start locateApp tutorial --type=ionic-angular

Dentro de la aplicación se ejecutará la comanda:
$ ionic serve -l

Esta comanda ejecutará la aplicación en un navegador donde se verán en 3 diferentes sistemas operativos: Windows, iPhone y Android, tal y como se puede ver en la ***Figura 9***. La aplicación a tratar será para Android, por lo que se centrará en esta aplicación por encima de los demás.

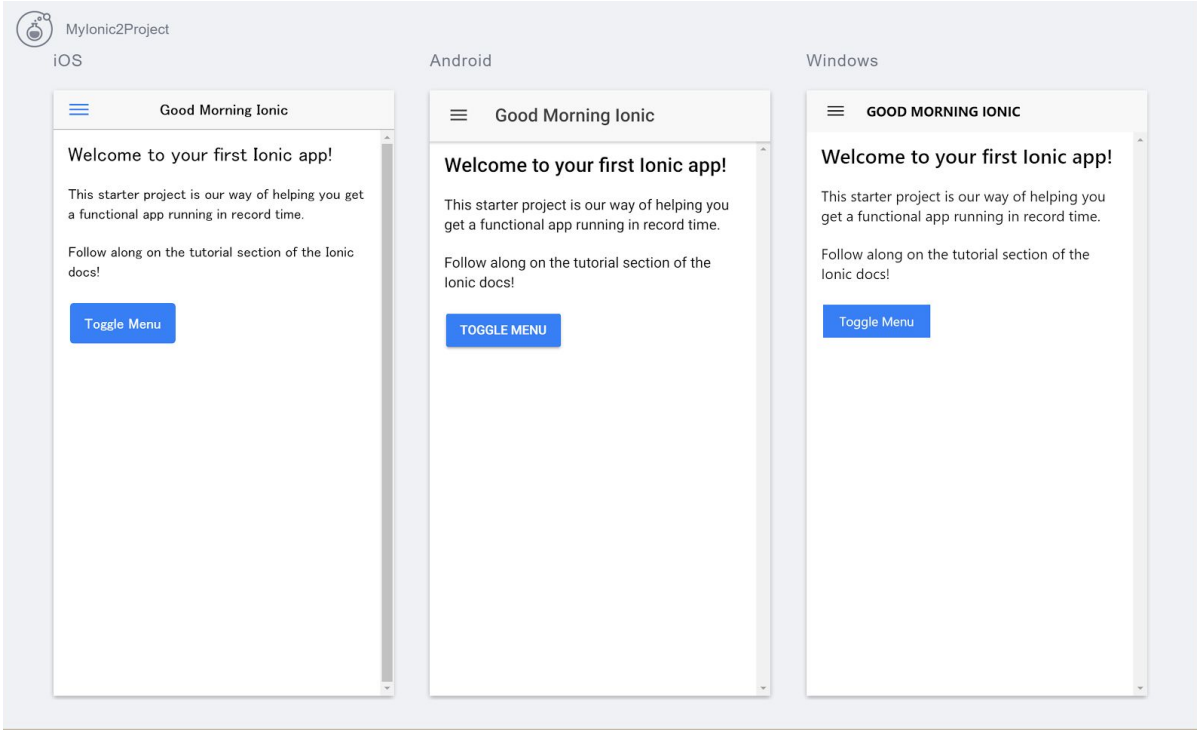

Figura 9. Ejecución de la aplicación en distintos Sistemas Operativos.

Una vez realizado esto, cualquier cambio que se realice en el código será reflejado automáticamente en la aplicación del navegador.

Ahora se pasa a editar el código. Para editar primero hemos de diferenciar entre 3 cosas importantes: controlador, vista y modelo.

El *modelo* representa la información con la que se trabaja; es decir, gestiona los accesos a la información. El *modelo* envía a la *vista* la información solicitada. Las peticiones llegan al *modelo* a través del *controlador*.

El *controlador* responde a los eventos realizados en la *vista* e invoca peticiones al *modelo* cuando se solicita determinada información.
La *vista* presenta el *modelo* en un formato adecuado; es decir, representa la información de la manera deseada.

Sabiendo como funciona la relación *modelo-vista-controlador*, que se puede reflejar mejor en la ***Figura 10***, pasamos a reconocer el modelo, la vista y el controlador.

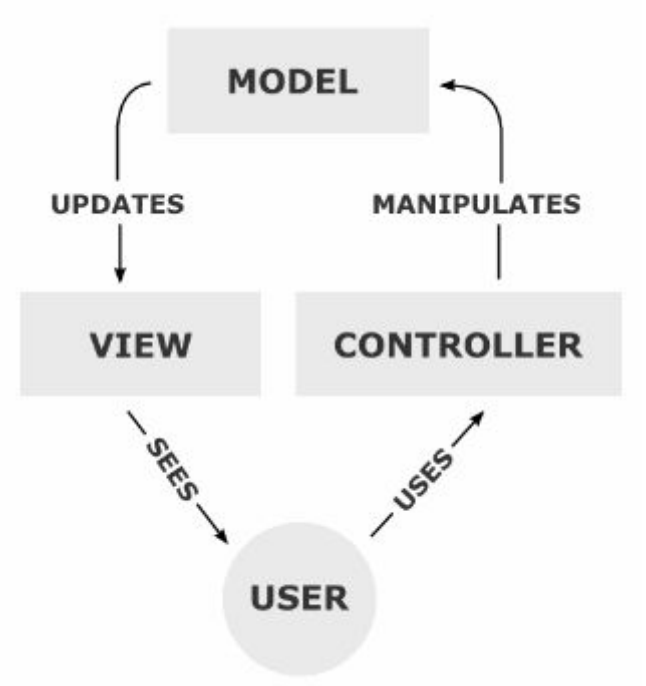

Figura 10. Típica relación Model-View-Controller.

El controlador, en este caso, se encontraría en la carpeta *page*, que sería la página a editar; en concreto, el archivo *.ts*. La vista sería el archivo *html*, ubicado en esta misma carpeta, y el modelo se hallaría en la carpeta *models* (si no existe, se ha de crear) y sería el archivo *.ts*. Si se desea mejorar la vista se puede editar el archivo *scss*, ubicado en la misma carpeta *page*, que equivaldría al *css* visto en cada página web.

Como se van a obtener datos del servidor C9, se ha de crear/editar un archivo que recupere estos datos. Este se creará/editará en la carpeta *models*. El archivo en esta carpeta se

llamará *section.ts* y trabajará con la información recopilada del servidor, un diccionario que guarda las potencias medias recibidas por sección y por Punto de Acceso.

Ahora creamos los archivos necesarios para la aplicación; también modificaremos lo que se creó con la opción *tutorial* y que no se necesitará para esta aplicación.

En la carpeta *pages* se creará la carpeta *home*, y dentro se crearán los archivos *home.html* y *home.ts*. Ahora tenemos la *vista* y el *controlador* correspondiente a la aplicación. También hemos de cambiar la página de inicio para que sea este nuevo archivo. Por eso, entraremos a la carpeta *app*, abriremos el archivo *app.module.ts* y cambiaremos la línea que diga *rootPage*, donde la página de inicio que una vez fue *HelloIonic* (creado por el propio tutorial) será cambiado por *HomePage*.

Aprovechando que estamos en este archivo eliminaremos las cosas que no necesitamos. Líneas que refieran a *helloionic*, *list* e *itemdetails* serán eliminados. Hay que eliminar los *import* de esas páginas ubicados en la parte superior, y también eliminarlos de las líneas de *entrycomponents* y *declarations*. A su vez, hay que añadir *home*, la nueva página a utilizar de inicio, a *import*, *entrycomponents* y *declarations*. Así mismo, se borrará *openPage* ya que no hay menú para empezar, y se modificará *this.page*, donde se borrará toda referencia a *helloionic*, *list* e *itemdetails* y se añadirá *HomePage*, la referencia a *home* ya que solo habrá una página.

Tener en cuenta que el formato a utilizar es el mismo que los que se acaban de eliminar; dicho esto, sería mejor revisar el formato previo, añadir *home* donde corresponde y luego borrar lo mencionado anteriormente.

También se puede quitar el *menuController* ya que no habrá menú en esta aplicación.

Una vez finalizado las modificaciones, se eliminarán las carpetas y archivos referentes a *helloionic*, *list* e *itemdetails*, así no nos dará error al momento de guardar.

Ahora cuando se pase a ver la aplicación, se verá que no hay nada debido a que aún no se ha modificado el archivo *home.ts*, y también desaparecerán los menú y páginas que inicialmente estaban. Es momento de pasar a editar código.

Primero se creará el plano donde se identificarán las 7 secciones.

El plano se creó a base de la etiqueta <>. Esta etiqueta hace referencias a un *menu* en muchas ocasiones, pero esta vez se utilizó para crear una sección del plano. Por eso, se crearon tantas etiquetas fueran posibles para asemejarse al plano real. Para hacer que todo tenga un aspecto correcto (que no se muestre el contorno por ejemplo) se ha de retocar el css de la página.

Una vez creado el plano también se han de asignar clases a estos segmentos. Esto se realiza para que cuando se quiera cambiar de color una sección del plano se pueda realizar de manera cómoda. Es decir, podemos decir que la sección grande, 6, que contiene varios

elementos de esta etiqueta cambie de color de blanco a verde para indicar el posicionamiento del terminal solo diciendo que cambie la clase *clase6* y esto cambiará de color todas las etiquetas correspondientes a esta clase.

Una vez finalizado la creación del mapa, se ha de crear el botón *Get Location*.

La finalidad del botón *Get Location* es de ejecutar la acción de recopilar datos de todos los puntos de accesos visibles desde el terminal, que posteriormente se utilizarán para el cálculo de la distancia Euclidiana junto a los datos recopilados del servidor. Estos datos se enviarán al servidor a través de una URL con todos los parámetros correspondientes, y esperará respuesta por parte del servidor.

Respecto a la URL a donde se van a enviar los datos, esto se realiza creando un Proveedor. Este Proveedor normalmente se ejecuta en el constructor de la clase principal de la aplicación, pero éste se ejecutará después de recibir los datos, cosa que facilita el manejo de información.

La respuesta del servidor será la sección a la cual el terminal pertenece, ya que tendrá la mínima distancia respecto a los demás, lo que conlleva que en la aplicación la sección perteneciente al plano creado cambie de color: de blanco a verde.

Esta acción se realizará solo una vez. Cada vez que se decidan hacer pruebas se volverá a apretar el botón *Get Location* y se volverá a repetir toda esta acción, dando por resultado la misma o una nueva sección.

Para probar la aplicación basta con abrir *Genymotion* y luego, dentro de la carpeta de la aplicación, ejecutar en la terminal la comanda:
$ ionic cordova run android

Para compilarlo y descargarlo en el dispositivo Android, se ha de revisar el dispositivo para que esté autorizado con la comanda:
$ adb devices

Y luego compilarlo con la misma comanda vista anteriormente:
$ ionic cordova run android

No hay que olvidarse de importar cada clase/archivo que se cree, ya que sino el código, por bien implementado que esté, no funcionará.

Es importante también recalcar que es mejor probar el código directamente en la aplicación que en una máquina virtual creada, como el *Genymotion*, ya que algunas librerías no funcionan en estas máquinas debido a que solo interactúan con el mismo *localhost* del ordenador, y eso puede crear muchos problemas. Si se va a utilizar una librería es mejor primero leer bien sobre ella y averiguar si funcionaría en un emulador o no.

A continuación se adjuntará las partes importantes del código implementado en general.

loc-section.ts

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/map';

import {Section} from '../../models/section';

@Injectable()
export class LocSectionProvider {

  c9URL = 'https://cris-app-schraderleo.c9users.io/'

  constructor(public http: Http) {
    console.log('Hello LocSectionProvider Provider');
  }

  load(data: string): Observable<Section> {
    return this.http.get(`${this.c9URL}/?list=${data}`)
      .map(res => <Section>(res.json()))
  }

}
```

home.ts

```
import { Component } from '@angular/core';
import { Section } from '../../models/section';
import { LocSectionProvider } from '../../providers/loc-section/loc-section';

declare var WifiWizard;
var newlist2 = [];

@Component({
  selector: 'page-home',
  templateUrl: 'home.html',
})
export class HomePage {

  lsp: LocSectionProvider;
  section: Section;

  turnthisgreen(segment){
    var element = document.getElementsByClassName(segment) as
HTMLCollectionOf<HTMLElement>;
    for(var elem in element){
      element[elem].style.backgroundColor="green";
    }
  }
```

```
  listhandler(newlist){

    newlist2 = [];

    for(var element=0; element<newlist.length; element++){
      newlist2.push(JSON.stringify(newlist[element]["BSSID"]));
      newlist2.push(JSON.stringify(newlist[element]["SSID"]));
      newlist2.push(JSON.stringify(newlist[element]["level"]));
    };

    this.lsp.load(JSON.stringify(newlist)).subscribe(
      section => {this.section = section;}
    )

    var thissegment = "seg" + this.section;

    var elementHTML = document.getElementsByClassName(thissegment) as
HTMLCollectionOf<HTMLElement>;
    for(var elem in elementHTML){
      element[elem].style.backgroundColor="green";
    }

  }

  fail(e){
    alert("Failed"+e);
  }

  getScanResult(){
    WifiWizard.getScanResults(this.listhandler, this.fail);
  }

}
```

home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>My Locate App</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-item>

    <ion-segment class="map">
      <ion-segment-button class="seg1" id="seg1">
      </ion-segment-button>
      <ion-segment-button class="seg2">
      </ion-segment-button>
```

```
    <ion-segment-button class="seg4" id="seg4">
    </ion-segment-button>
  </ion-segment>

  <ion-segment class="map">
    <ion-segment-button class="seg1" id="seg12">
    </ion-segment-button>
    <ion-segment-button class="seg3">
    </ion-segment-button>
    <ion-segment-button class="seg4" id="seg42">
    </ion-segment-button>
  </ion-segment>

  <ion-segment>
    <ion-segment-button class="seg5" id="seg5">
    </ion-segment-button>
    <ion-segment-button class="seg6" id="seg6">
    </ion-segment-button>
    <ion-segment-button class="seg6" id="seg62">
    </ion-segment-button>
  </ion-segment>

  <ion-segment>
    <ion-segment-button class="seg5" id="seg52">
    </ion-segment-button>
    <ion-segment-button class="seg6" id="seg63">
    </ion-segment-button>
    <ion-segment-button class="seg6" id="seg64">
    </ion-segment-button>
  </ion-segment>

  <ion-segment>
    <ion-segment-button class="seg7">
    </ion-segment-button>
    <ion-segment-button id="segdes1">
    </ion-segment-button>
    <ion-segment-button id="segdes2">
    </ion-segment-button>
  </ion-segment>

  <ion-segment >
    <ion-segment-button id="seg7">
    </ion-segment-button>
    <ion-segment-button id="segdes3">
    </ion-segment-button>
    <ion-segment-button id="segdes4">
    </ion-segment-button>
  </ion-segment>
</ion-item>
```

```
    <button ion-button full (click)="getScanResult(this)">Get location</button>

</ion-content>
```

home.css

```
ion-segment-button{ border-style: solid; border-color: blue;}

#seg12, #seg42, #seg52, #seg63, #seg64, #segdes3, #segdes4{border-top-style: none;}
#seg1,#seg4, #seg62, #seg5, #seg6, #segdes1, #segdes2, #seg7, #segdes3,
#segdes4{border-bottom-style: none}
#segdes1, #segdes2, #segdes3, #segdes4, #seg7, #seg6, #seg63{border-left-style:none;
border-right-style:none;}
#seg4, #seg42, #seg62, #seg64{border-left-style:none}
#seg1, #seg12{border-right-style:none}
```

12. VERIFICACIÓN

Una vez implementado el código, se ha de verificar el correcto funcionamiento de la aplicación.

El código que recopila los datos, ver ***Figura 11***, así como el que saca las potencias medias y los divide por Puntos de Accesos para obtener la base de datos final, ver ***Figura 12***, funcionan. Este paso era importante que funciona ya que de esta base de datos dependerá el resultado final de la aplicación. Cabe recalcar que las figuras vistas debajos son ejemplos realizados antes de la captura buena de datos, que contiene datos adquiridos durante 5 minutos de escaneo y para las 7 secciones correspondientes.

```
1  2017-06-17 18:35:11,'62:02:71:75:89:24',-91
2  2017-06-17 18:35:11,'98:97:D1:0D:AC:F3',-77
3  2017-06-17 18:35:11,'D4:7B:B0:09:18:88',-47
4  2017-06-17 18:35:13,'62:02:71:75:89:24',-92
5  2017-06-17 18:35:13,'98:97:D1:0D:AC:F3',-73
6  2017-06-17 18:35:13,'D4:7B:B0:09:18:88',-42
7  2017-06-17 18:35:15,'62:02:71:75:89:24',-93
8  2017-06-17 18:35:15,'98:97:D1:0D:AC:F3',-71
9  2017-06-17 18:35:15,'D4:7B:B0:09:18:88',-42
```

Figura 11. Ejemplo de recopilación de datos durante tiempo determinado de escaneo.

```
1  {'62:02:71:75:89:24': {'1': 88, '2_0': 92, '2_1': 91}}
2  {'98:97:D1:0D:AC:F3': {'1': 78, '2_0': 73, '2_1': 75}}
3  {'D4:7B:B0:09:18:88': {'1': 34, '2_0': 28, '2_1': 31}}
```

Figura 12. Ejemplo de potencias medias por sección adjuntadas por MAC de los AP's.

Para probar previamente el correcto funcionamiento del servidor, primero se ha debido de introducir datos "falsos" a la URL debida para ver qué responde el servidor a ello. Es por eso que se ha decidido enviar datos en el formato deseado. La URL completa a probar ha sido:
https://cris-app-schraderleo.c9users.io/hello.php?list=MOVISTAR_1887,D4:7B:B0:09:18:88,42,MOVISTAR_BE98,98:97:D1:0D:AC:F3,40,MOVISTAR_8871,D4:7B:B0:79:48:88,49,MOVISTAR_AAAA,62:02:71:75:89:24,60
Y la respuesta que da el servidor, con los datos de prueba, ha sido "2". La URL que se ha devuelto ha sido la siguiente:
https://cris-app-schraderleo.c9users.io/response.php?value=2

Y tal como se había dicho en el apartado de Implementación, el valor que aparece en la etiqueta *value* es el que se muestra en la página *response.php* tal y como se puede apreciar en la ***Figura 13***.

Figura 13. Respuesta del servidor la acción realizada en hello.php.

Por parte de la aplicación, la primera prueba que se realizó fue cambiar de color una sección en particular del plano. Debido a que la sección 6 es grande, se decidió cambiar el color a la etiqueta *class* de valor *seg6*. Esto funcionaba.

Después se decidió probar la recopilación de datos. La librería *WifiWizard* dio problemas hasta que se vio que trabajaba correctamente en un dispositivo real y no en un emulador/máquina virtual, en donde solo devolvía los datos correspondientes a la red conectada. Esto se pudo ver compilando la aplicación a un dispositivo Android y creando una alerta que muestre los datos que recopila. Esto también funcionaba.

Después se decidió enviar datos al servidor. La prueba fue simple ya que se cambió el *print* del ejecutable *execpy.py* para que devolviera *1* en todas las ocasiones, y con un *alert* en la aplicación se podía ver este *1*. Otra forma de comprobarlo es accediendo a los *logs* del servidor donde aparecen los intentos de conexión al servidor.

Como el programa se ha ido probando por partes, solo quedaba ejecutar todo el código completo.

La prueba final se realizó en la sección 6, después de recopilar todos los datos necesarios. La manera de recopilar estos datos en el mapa se puede reflejar en la ***Figura 14***.

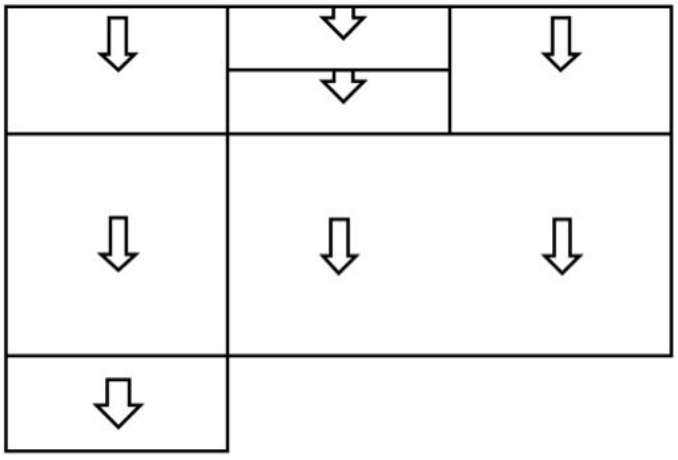

Figura 14. Puntos de recopilación de datos en el plano.

Debido a que la sección 6 era la más grande, se hicieron 2 recopilaciones de datos y se realizó la prueba en esta sección. Una vez obtenido los datos, se dispuso a localizar el dispositivo con la aplicación. El resultado se puede apreciar en la ***Figura 15***, donde la sección que cambia de color es, en efecto, la perteneciente a la sección 6.

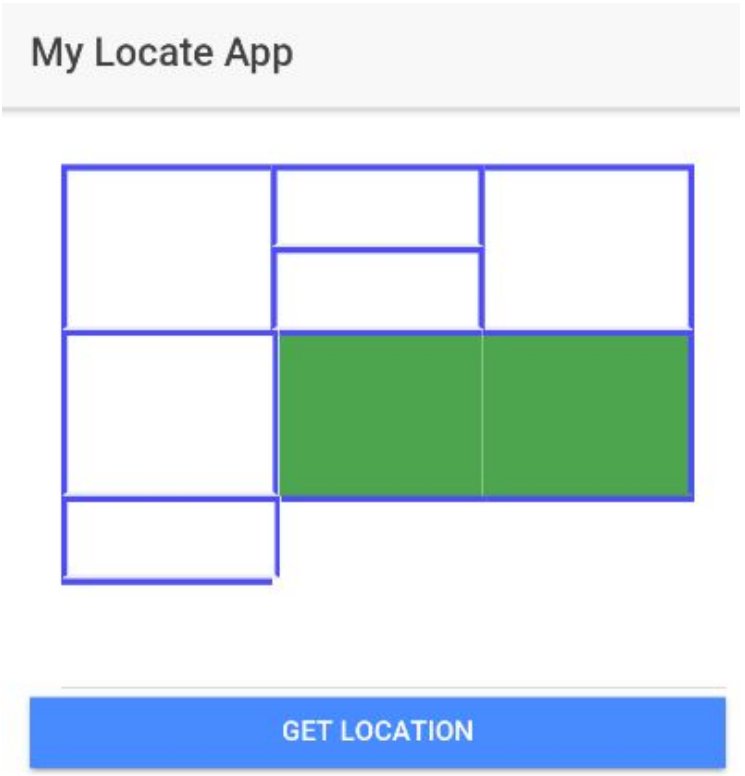

Figura 15. Localización del dispositivo en la sección 6.

13. CONCLUSIONES

El proyecto en general ha sido interesante a la hora de desarrollarlo.

Han habido 3 algoritmos que se han podido implementar con distintas técnicas de localización. El uso de la tecnología Wifi solo se ha debido a su uso cotidiano, por lo que en otras situaciones incluso se ha podido llegar a emplear dispositivos como una Raspberry. El manejo de datos se ha podido incluso desarrollar con MySQL o alguna otra base de datos y ahorrarnos el uso de varios archivos. El servidor incluso se ha podido crear desde cero y trabajar con Apache creando una estructura similar a la trabajada ya durante el Grado. La aplicación se ha podido realizar en otro Framework que pudo haber resultado más fácil, y se pudo haber implementado los cálculos incluso dentro de la aplicación.

Hay muchas maneras en las que algo se puede implementar, pero el hecho de haberlo implementado de esta manera ha tenido sus pro y sus contras.

Por parte de los pros, cabe destacar:

-El uso del lenguaje Python, un lenguaje de alto nivel y fácil de implementar a la hora de desarrollar los códigos, y como es un lenguaje muy usado hay respuesta para toda pregunta en Internet.
-El uso del algoritmo y la técnica de localización desarrollada en este proyecto han sido de las más fáciles de implementar.
-El uso de Cloud9 aprovechando una cuenta ya existente, aunque actualmente se ha de registrar con la tarjeta de crédito/débito para crear una cuenta incluso si es la simple y sin coste alguno.
-El uso de Ionic 2, un Framework cuyo desarrollo es muy parecido al de Web2py (uno utilizado en una asignatura del Grado) e hizo que la implementación de código se desarrolle de manera natural.
-El aprendizaje final de cosas como Ionic 2, PHP, TypeScript y algoritmos en interiores para localizar dispositivos, un tema muy amplio y que ha sido divertido explorarlo.

Por parte de las contras, cabe resaltar por encima de todo Ionic 2.

Ciertamente es un Framework muy potente y te permite desarrollar aplicaciones para Android, iPhone y Windows de manera paralela pero hay muchas librerías que dependen de un dispositivo real para su funcionamiento correcto. Como uno trabaja en un emulador o una máquina virtual para probar el código, normalmente se trabaja con las especificaciones dadas por ello. Casos como la librería WifiWizard dan problemas porque uno puede implementarlo de la manera correcta pero aún así no saber qué falla. Me costó una semana dar con el problema de que no trabaja correctamente con Genymotion porque no da los resultados deseados (todos los puntos de accesos que detecta un ordenador normal). Otro caso similar fue el de los Proveedores.

Y también trabajar con objetos fue un dolor de cabeza. La manera en la que se reconocen los objetos da problemas al momento de hacer pruebas. Un simple *alert()* de un objeto que recibes a veces has de pasarlo específicamente a string (*JSON.stringify()*) unas 2 veces para que funcione. Cosas que no llegué a entender del todo pero que, de una u otra manera, salieron adelante.

Aún así, el sistema implementado funciona correctamente y será útil siempre que se trabaje en el plano planteado. Se podría decir que los objetivos han sido alcanzados, aunque algunos han costado más que otros. La arquitectura para conseguir los datos base ha sido clara y concisa y solo depende de un ordenador. La subida de datos al servidor, siendo de manera manual, puede que no sea la más adecuada pero es algo que no afecta al desarrollo total del proyecto. Los cálculos en el servidor facilita que la aplicación no descargue datos extra y trabaje de una manera óptima. La aplicación presenta la interfaz deseada y trabaja como debe.

La finalización ha sido la deseada, aunque siempre queda margen de mejora.

Es una lástima que al finalizar el proyecto vea mejoras que se han podido implementar, como crear un menú del cual una opción facilite la creación de tu propio plano del edificio, o importar uno y pasarlo al formato deseado. Otra cosa que no se implementó fue poder pasar desde tu aplicación el archivo de datos base con el que trabajarás, el que está subido previamente a Cloud9, o implementar una página donde puedas subir tus archivos y personalizarlos, entre otros. Pero todo esto visto desde una aplicación hecha a medida para cualquier situación, ya que la aplicación actual solo está diseñada para el plano en donde se ha trabajado.

Por otro lado, de cara al futuro y dejando de lado las posibles mejoras de la aplicación, si se quiere implementar el mismo sistema en un plano grande probablemente requiera aplicar más técnicas de localización para combatir el efecto multipath, así mismo como aumentar la captura de datos en una sección para poder mejorar la precisión.

Por último, cabe recalcar que los conocimientos adquiridos durante la carrera fueron de gran ayuda al momento de desarrollar la aplicación, ya que cuando uno está familiarizado con algo todo fluye mejor, y eso fue el caso de herramientos como *Python*, *iwlist* entre otros.

14. BIBLIOGRAFÍA

[1] G. Mao, B. Fidan, B. Andersonels. “Wireless sensor network localization techniques", en *Computer networks, vol. 51, no. 10, pp. 2529-2553, November 2007*.

[2] Yilin Zhao, Motorola Inc. “Standarization of Mobile Phone Positioning for 3G Systems” en *Standard Reports, IEEE Communications Magazine, July 2002*.

[3] P. Prasithsangaree, P. Krishnamurthy y P.K. Chrysanthis. “On indoor position location with wireless lans”, en *13th IEEE Int'l Symposium on Personal, Indoor, and Mobile Radio Communications, 2002*.

[4] David Sánchez, Sergio Afonso, Elsa M. Macías, Member IAENG y Álvaro Suárez. “Devices Location in 802.11 Infrastructure Networks using Triangulation ”, en *IMECS 2006*.

[5] Eiman Elnahrawy, Xiaoyan Li y Richard P. Martin. “The Limits of Localization Using Signal Strength: A Comparative Study ”, en *Sensor and Ad Hoc Communications and Networks, 2004*.

[6] Dhruv Pandya, Ravi Jain, E. Lupu. “Indoor location estimation using multiple wireless technologies”, en *Personal, Indoor and Mobile Radio Communications, 2003*.

[7] Documentación Ionic
http://ionicframework.com/docs/

[8] Localización en Interiores, Artículo en Bit
https://pablogomezoviedo.wordpress.com/2014/07/22/localizacion-interiores-articulo-bit/

[9] Documentación Cloud9
https://docs.c9.io/docs/